Programmiersprachen
Sommersemester 2017

Aufgabensammlung 3

Die Aufgaben werden am **22. Mai** in der Übung bewertet. Diese Aufgabensammlung beschäftigt sich mit den Grundlagen zu Templates, der Standard Template Library (STL) und Lambdas.
Testen Sie alle entwickelten Datentypen, Methoden und Funktionen. Achten Sie auf `const`-Korrektheit. Legen Sie für jede Klasse eine *hpp* und eine *cpp* im Ordner `source` an und erweitern Sie entsprechend die Datei `source/CMakeLists.txt`. Verwenden Sie zur Initialisierung der Membervariablen immer die Member-initialization-list. Die letzten beiden Aufgaben sind Optional. Nutzen Sie neben dem Vorlesungsskript ausschließlich aktuelle Fachliteratur oder Online-Referenzen, z.B.

- Stroustrup, B.: Einführung in die Programmierung mit C++ (2010)
- http://en.cppreference.com/
- http://www.cplusplus.com/

Bei Fragen und Anmerkungen schreiben Sie bitte ein Email an andreas.bernstein@uni-weimar.de .

Aufgabe 3.1

Erstellen Sie unter ihrem eigenen *github*-Account ein neues Repository mit dem Namen *programmiersprachen-aufgabenblatt-3*. Klonen Sie das erstellte Repository. Erstellen Sie darin die gleiche Struktur wie im *programmiersprachen-aufgabenblatt-2*-Repository. Vergessen Sie die Datei `.gitignore` nicht.
Sie können auch ihr Repository von Aufgabenblatt 2 weiter verwenden.
Fügen Sie folgendes Beispielprogramm hinzu und kompilieren Sie es.

```
#include <cstdlib>    // std::rand()
#include <vector>     // std::vector<>
#include <list>       // std::list<>
#include <iostream>   // std::cout
#include <iterator>   // std::ostream_iterator<>
#include <algorithm> // std::reverse, std::generate

int main()
{
```

```
  std::vector<int> v0(10);

  for (auto& v : v0) {
    v = std::rand();
  }
  std::copy(std::begin(v0), std::end(v0),
            std::ostream_iterator<int>(std::cout, "\n"));

  std::list<int> l0(v0.size());
  std::copy(std::begin(v0), std::end(v0), std::begin(l0));

  std::list<int> l1(std::begin(l0), std::end(l0));
  std::reverse(std::begin(l1), std::end(l1));
  std::copy(std::begin(l1), std::end(l1),
            std::ostream_iterator<int>(std::cout, "\n"));

  l1.sort();
  std::copy(l1.begin(), l1.end(),
            std::ostream_iterator<int>(std::cout, "\n"));

  std::generate(std::begin(v0), std::end(v0), std::rand);
  std::copy(v0.rbegin(), v0.rend(),
            std::ostream_iterator<int>(std::cout, "\n"));

  return 0;
}
```

Analysieren Sie das Programm und erläutern Sie dessen Funktionsweise. Benennen Sie die *Typen* aller Variablen. **[5 Punkte]**

Aufgabe 3.2

Erstellen Sie ein neues Programm mit dem Namen `aufgabe2bis4.cpp` und fügen Sie es zur Datei `source/CMakeLists.txt` hinzu.
Instanziieren Sie eine `std::list` mit `unsigned int` und füllen Sie diese mit 100 Zufallszahlen von 0 bis 100. Erzeugen Sie einen `std::vector` und kopieren Sie mit `std::copy` die Elemente der Liste in den `std::vector`. **[5 Punkte]**

Aufgabe 3.3

Bestimmen Sie, wieviele unterschiedliche Zahlen in der `std::list` aus Aufgabe 3.2 sind und geben Sie die Zahlen von 0 bis 100 aus, die *nicht* in der Liste

sind. Verwenden Sie `std::set` für ihre Lösung. **[5 Punkte]**

Aufgabe 3.4

Ermitteln Sie die Häufigkeit jeder Zahl in der `std::list` aus Aufgabe 3.2. Erklären Sie, warum sich `std::map` für dieses Problem anbietet? Geben Sie die Häufigkeit aller Zahlen in der Form *Zahl : Häufigkeit* auf der Konsole aus. **[8 Punkte]**

Aufgabe 3.5

Erstellen Sie ein neues Programm `aufgabe5.cpp`.

```
#define CATCH_CONFIG_RUNNER
#include <catch.hpp>
#include <cmath>
#include <algorithm>

TEST_CASE("filter alle vielfache von drei", "[erase]")
{
  // ihre Loesung :
  // ...

  REQUIRE(std::all_of(v.begin(), v.end(), is_multiple_of_3));
}

int main(int argc, char* argv[])
{
  return Catch::Session().run(argc, argv);
}
```

Füllen Sie einen `std::vector` von `unsigned int` mit 100 Zufallszahlen von 0 bis 100. Entfernen Sie alle Zahlen, die nicht durch drei teilbar sind. Lesen Sie dazu folgenden Wikipedia-Eintrag: https://en.wikipedia.org/wiki/Erase-remove_idiom Testen Sie danach mit `std::all_of` aus `<algorithm>`, ob alle Elemente im vector vielfache von drei sind. Schreiben Sie dafür eine Hilfsfunktion `is_multiple_of_3`. **[5 Punkte]**

Aufgabe 3.6

Erklären Sie die Unterschiede zwischen *sequentiellen* und *assoziativen* Containern. Wählen Sie für folgende Anwendungsfälle einen Container und erklären

Sie ihre Wahl:

- Speichern der Punkte eines Polygons
- Zuordnung von Farbnamen und entsprechenden RGB-Werten
- FIFO-Warteschlange von Druckaufträgen

[5 Punkte]

Aufgabe 3.7

Erstellen Sie ein neues Programm. Am besten verwenden Sie das Codefragment aus Aufgabe 3.5.
Objekte der Klasse `Circle` sollen in einem `std::vector` gespeichert und der Radiusgröße nach sortiert werden. Um Objekte in einem Container sortieren zu können, müssen Sie vergleichbar sein. Überladen Sie die Operatoren `operator<`, `operator>` und `operator==` für Objekte vom Typ `Circle`, füllen Sie einen Container mit Objekten vom Typ `Circle` und sortieren Sie diesen.

```
REQUIRE(std::is_sorted(container.begin(), container.end()));
```

[6 Punkte]

Hinweis: Sie können die `draw`-Methoden und den Header `window.hpp` aus `Circle.hpp/.cpp` entfernen. Dann müssen Sie nicht gegen `glfw` und `${GLFW_LIBRARIES}` linken.

Aufgabe 3.8

Objekte der Klasse `Circle` sollen in einem STL-Container gespeichert und der Radiusgröße nach sortiert werden. Diesmal soll allerdings `std::sort` die Vergleichsfunktion als Parameter übergeben werden. Implementieren Sie die Vergleichsfunktion mit Hilfe eines Lambdas. Testen Sie danach mit der Funktion `std::is_sorted` ob der Container sortiert ist.

```
REQUIRE(std::is_sorted(container.begin(), container.end()));
```

[4 Punkte]

Aufgabe 3.9

Addieren Sie die gegeben Container `v1` und `v2` elementweise auf und speichern Sie das Ergebnis im Container v3. Verwenden Sie dafür den Algorithmus `std::transform` und ein Lambda.

```
std::vector<int> v1{1,2,3,4,5,6,7,8,9};
std::vector<int> v2{9,8,7,6,5,4,3,2,1};
std::vector<int> v3(9);
```

Testen Sie danach mit `REQUIRE( std::all_of( . . .`, dass die Elemente in `v3` alle gleich zehn sind unter Verwendung eines Lambdas. **[5 Punkte]**

Aufgabe 3.10

Schreiben Sie ein Funktionstemplate `filter`, welches als ersten Parameter einen sequentiellen Container und als zweiten Parameter ein Prädikat hat. Die Funktion soll einen neuen Container gleichen Typs zurückgeben. Dieser soll nur Werte enthalten, die das Prädikat erfüllen. Verwenden kann man die Funktion dann wie im folgenden Beispiel.

```
std::vector<int> v{1,2,3,4,5,6};
std::vector<int> alleven = filter(v, is_even);
```

mit

```
bool is_even(int n) { return n % 2 == 0; }
```

Testen Sie ihre Funktion. Sie können sich am Codefragment aus Aufgabe 3.5 orientieren, um ein Testprogramm zu schreiben. **[6 Punkte]**

Aufgabe 3.11

Erklären Sie, warum es bei folgendem Programmsegment zu Problemen kommen kann:

```
std::map<string,int> matrikelnummern;
//Hinzufueggen von vielen Studenten
matrikelnummern["Max Mustermann"] = 12345;
matrikelnummern["Erika Mustermann"] = 23523;
//...
exmatrikulation(matrikelnummer["Fred Fuchs"]);
```

Wie vermeidet man diese Problem? Welche Möglichkeiten gibt es denn zum Einfügen und zum Suchen? **[5 Punkte]**

Hinweis: Welche Suchmethoden sind `const`?

Aufgabe 3.12

Legen Sie einen `std::vector` mit Objekten der Klasse `Circle` an. Alle Kreise sollen verschiedene Radien haben. Zum Beispiel:

```
// Vorrausgesetzt es gibt einen Konstruktor
// der einen Radius als Parameter bekommt
std::vector<Circle> circles{{5.0f},{3.0f},{8.0f},
                            {1.0f},{5.0f}};
```

Kopieren Sie anschliessend mit dem Algorithmus `copy_if` alle Kreise deren Radius größer als 4.0f ist, in einen zweiten `std::vector`. Verwenden Sie für das benötigte Prädikat wieder ein Lambda. Testen Sie danach mit `std::all_of`, dass die Radien im Zielcontainer alle größer drei sind (unter Verwendung eines Lambdas). **[5 Punkte, optional]**

Aufgabe 3.13

Lösen Sie Aufgabe 3.8 unter Verwendung eines Funktors. **[3 Punkte, optional]**

Bei Fragen und Anmerkungen schreiben Sie bitte ein Email an andreas.bernstein@uni-weimar.de .